Computer systems are a pervasive aspect of daily life. From the smart phones we use to access our favorite websites, to the services those websites provide, to the computer systems that manage core infrastructure services (e.g., air traffic control, the power grid, or the stock market), there are very few aspects of a normal day not mediated by computer systems. Two byproducts of this integration are increased frustration when these systems don't work properly and corresponding appreciation for the value of dependable systems.

The goal of my research is to build robust distributed systems that simply work despite the best, and worst, efforts of the environment. Building robust systems is challenging for multiple reasons. On the one hand, the definition of "works" is system dependent and frequently invokes a tension between performance and some notion of correct behavior. On the other hand, there are many environmental factors that can get in the way: crashed components, corrupted hardware, transient electromagnetic interference, delayed and corrupted messages, misconfigured servers, malicious attackers, disaffected sysadmins, selfish users, benign concurrency, changes in workload, ...

My approach is based on a synergy between system building and theoretical work. To be confident that my systems work as advertised, I base my design and implementation efforts on solid theoretical foundations. To ensure that my theoretical foundations are relevant, I leverage the system building experience to develop new theoretical models and correctness conditions that better match the system deployment requirements.

I have applied this methodology to a variety of applications in domains ranging from peer-to-peer backup to live video streaming to scalable cluster file systems. In the process I have developed new fault models and built systems that are both more practical and more robust than previously thought possible. In the following I briefly overview of my previous work and future plans.[1]

Previous and ongoing work

Making Byzantine Fault tolerant systems tolerate Byzantine faults. One major impediment to building robust distributed systems is arbitrary, or Byzantine, behavior of individual components. Systems robust to arbitrary failures are said to be Byzantine fault tolerant (BFT). In the context of BFT systems, our Aardvark system [14] demonstrated the benefits of redefining the standard performance specification of BFT systems. The goals of past BFT systems were defined in keeping with the spirt Lampson's classic advice, "The normal case must be fast. The worst case must make some progress." The result was a collection of fault tolerant systems which performed well in the absence of failures, but could be brought to their knees by a single fault. In retrospect, the reason for the discrepancy is obvious: an adversary can be relied upon to transform the worst case into the normal case! With Aardvark, we demonstrated that it is possible to design BFT systems that perform well in both the absence *and* presence of failures. Our other work on BFT systems includes the Zyzzyva replication library [16, 20, 24] which dramatically improved the throughput of BFT systems and BFT deployments of Zookeeper and HDFS (important infrastructure services in commercial datacenters) based on our UpRight replication library [13]. On the foundational front we have shown that eliminating equivocation does not simplify coordination [5]

[1] I use "we" when describing previous projects, to acknowledge the collaborative nature of the work.

and developed a refined failure model that bridges the gap between synchrony and asynchrony by more accurately capturing the requirements of datacenter environments [2].

Fault tolerance meets game theory. Our work on BAR fault tolerance pioneered the development of a new failure model and framework for designing systems that span multiple administrative domains (MADs). In MAD systems, like any distributed system, nodes can fail in arbitrary ways. Unlike traditional distributed systems, however, nodes may also misbehave because the user controlling the node selfishly seek to maximize his personal benefit. The system specification must be designed to accommodate both arbitrary and rational behaviors. Traditional Byzantine fault tolerance from the distributed computing literature is inapplicable in MAD systems—theoretically, selfish nodes could be modeled as Byzantine, but as more nodes behave selfishly the number of Byzantine nodes may well exceed the threshold above which most interesting distributed problem become provably impossible to solve. Traditional game theory from the economics literature is equally insufficient, since it does not account for the possibility that arbitrarily faulty nodes may behave irrationally. We addressed these challenges by introducing the BAR (**B**yzantine, **A**ltruistic, **R**ational) failure model as a framework for reasoning about MAD systems [11, 18, 27]. We leveraged the BAR model to build BAR-B [27], a peer-to-peer cooperative backup service, and Flightpath [21, 26], a peer-to-peer media streaming service. BAR-B and Flightpath demonstrate that it is possible to build practical MAD systems that are robust to both Byzantine and rational behaviors.

Sybil defenses. Many fault tolerant systems assume that some large fraction of components are correct. When we move to extremely large systems with open membership (e.g. peer-to-peer systems, social networks, etc.) this assumption may no longer hold—an adversary who creates multiple accounts is known as a Sybil attacker. Our work in this area has identified a key distinction between two strategies for building systems that are robust to Sybil attacks. While the literature has historically grouped all systems that defend against Sybil attacks under the label of "Sybil defense" and freely compared them to each other, we believe that there is a fundamental difference between *Sybil detection* systems designed to identify Sybil users and *Sybil tolerant* systems designed to be robust to Sybil behavior [9]. Recognizing this distinction helped us to identify a credit network-based design pattern common to previous Sybil tolerant systems, which we were able to leverage to build Genie [8], a novel system for rate-limiting crawlers in online social networks. We have also identified a fundamental relationship between Sybil detection protocols and the theory of random walks at the foundation of web search [3, 4].

Consistency in geo-replicated systems. In the previously discussed projects, replication was used to ensure correct system behavior despite faulty components; in geo-replicated environments replication provides site diversity and enables low latency responses, provided that eventually consistent operation (i.e., short term replica divergence) is acceptable. Previous eventually consistent systems have suffered from two dramatic limitations. First, in order to ensure eventual replica convergence they have required history to be rewritten or required *all* operations to commute. Second, even when convergence has been achieved the end-to-end behavior of the system is rarely correct. Our work on Gemini [7], a replicated transaction processing system with write-anywhere semantics, makes two key contributions. First, we observe that the current state of a replica is a key source of non-determinism that must be accounted for in order to ensure eventual convergence. Second, we explicitly identify situations when eventual consistency is compatible with correct system behavior and situations when immediate replica coordination is required. We have also developed Depot [12], a distributed storage service that leverages eventual consistency to tolerate

Byzantine behavior at minimal cost. The key insight that allows Depot to work is simple: when data is augmented with a modification history and authenticated, the most dangerous Byzantine behaviors are indistinguishable from a pair of correct nodes that act concurrently.

Big data processing. The explosion of "big data" problems has led to an abundance of data processing frameworks, each tuned to specific assumptions on the computation structure and properties of the input data. One consequence of this diversity is that small changes to the computation being done or to the input data may necessitate switching to a different computation framework and/or model. Our Musketeer system [1] treats each framework as a microkernel and compiles each job into a generic intermediate representation that is automatically executed by the optimal framework(s).

Future Work

I envision a future where the distributed and networked systems that form the backbone of our computing infrastructure "just work." While we have made significant progress towards masking faulty components and misbehaving users, significant work remains to fully mask the impact of these threats. As computer systems become more pervasive, additional challenges beyond the traditional threats are emerging: distributed and networked systems are increasing in scale to include more diverse components in more diverse locations, resources are increasingly shared by multiple systems, and existing systems are being re-purposed to support new services and workloads. My long term research agenda is to rise to the challenges posed and develop the implementation and theoretical infrastructures to ensure that future distributed and networked systems "just work." Some near-to-intermediate term plans follow.

Adapting to changes in workload. Server resources an either be provisioned statically to support an expected maximal load or dynamically to respond to workload fluctuations. In both cases, heavy activity spikes can lead to significant degradations of performance and the appearance of a system that is no longer working as intended. In the former case, the cross-request interference of a heavy activity burst can prevent any user requests from completing in a timely fashion. Inspired by network congestion control techniques, I believe that strategic request triage in which selected requests are eliminated early can be an effective tool for ensuring that a maximal set of requests are processed promptly and providing graceful service degradation. In the latter case, while it is easy to add new resources to handle traffic bursts marked by heavy reads and writes to disjoint objects, standard techniques are inapplicable when the burst of writes are applied to the same non-partitionable object or database row. I plan to leverage eventual consistency as a tool to support dynamic re-provisioning of resources, in contrast to its standard use to mask the impact of high latency communication.

Network and system virtualization. Virtualization is a powerful tool that allows organizations to amortize the cost of purchasing and maintaining a computing infrastructure across multiple users. The promise of machine virtualization, and SDN based network virtualization, is that users are able to acquire and release resources as needed and that the acquired resources will be indistinguishable from a physical cluster that never fails. Current reality falls short: interference from other tenants makes performance unpredictable and the virtual cluster as a whole is susceptible to failure of specific components. I plan to leverage the synergy between virtual machines and SDN to provide reliable cluster virtualization infrastructure, applying fault tolerant replication to the SDN infrastructure and network-based resource management to ensure isolation as needed.

Allen Clement — Publication List

[1] I. Gog, N. Crooks, M. Schwarzkopf, M. Grosvenor, A. Clement, and S. Hand. Musketeer: strength in framework diversity. *Under submission*.

[2] D. Porto, C. Li, J. Leit ao, A. Clement, F. Junquiera, and R. Rodrigues. Visigoth fault tolerance. *Under submission*.

[3] L. Alvisi, A. Clement, A. Epasto, S. Lattanzi, and A. Panconesi. Communities, random walks, and social sybil defense. *Internet Mathematics*, to appear.

[4] L. Alvisi, A. Clement, A. Epasto, S. Lattenzi, and A. Panconesi. SoK: The evolution of sybil defense via social networks. In *IEEE Symposium on Security and Privacy (OAKLAND)*, 2013.

[5] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues. On the (limited) power of non-equivocation. In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2012.

[6] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin. EVE: Execute-verify replication for multi-core servers. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[7] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguica, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[8] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. Gummadi, A. Mislove, and A. Post. Defending against large-scale crawls in online social networks. In *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2012.

[9] B. Viswanath, M. Mondal, A. Clement, P. Druschel, K. Gummadi, A. Mislove, and A. Post. Exploring the design space of social network-based Sybil defense. In *International Conference on Communication Systems and Networks (COMSNETS)*, 2012.

[10] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. *ACM Transactions on Computer Systems (TOCS)*, 2011.

[11] E. Wong, I. Levy, L. Alvisi, A. Clement, and M. Dahlin. Regret-freedom isn't free. In *International Conference on Principles of Distributed Systems (OPODIS)*, 2011.

[12] P. Mahajan, S. Setty, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[13] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riché. Upright cluster services. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[14] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[15] F. Mari, I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, and H. C. Li. Model checking coalition nash equilibria in MAD distributed systems. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2009.

[16] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Transactions on Computer Systems (TOCS)*, 2009.

[17] A. S. Aiyer, L. Alvisi, R. Bazzi, and A. Clement. Matrix signatures: From MACs to digital signatures in distributed systems. In *International Symposium on Distributed Computing (DISC)*, 2008.

[18] A. Clement, H. C. Li, J. Napper, J.P. Martin, L. Alvisi, and M. Dahlin. BAR primer. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2008.

[19] A. Clement. Distributed computing in SOSP and OSDI. *SIGACT News*, 2008.

[20] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. *Communications of the ACM (CACM)*, 2008.

[21] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. Flightpath: Obedience vs. choice in cooperative services. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[22] F. Mari, I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, and H. C. Li. Model checking nash equilibria in MAD distributed systems. In *Formal Methods in Computer Aided Design (FMCAD)*, 2008.

[23] A. Clement, J. Napper, H. Li, J.P. Martin, L. Alvisi, and M. Dahlin. Brief announcement: Theory of BAR games. In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2007.

[24] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[25] H. C. Li, A. Clement, A. S. Aiyer, and L. Alvisi. The Paxos register. In *IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2007.

[26] H. C. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR Gossip. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[27] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2005.